

|  |
| --- |
|  |

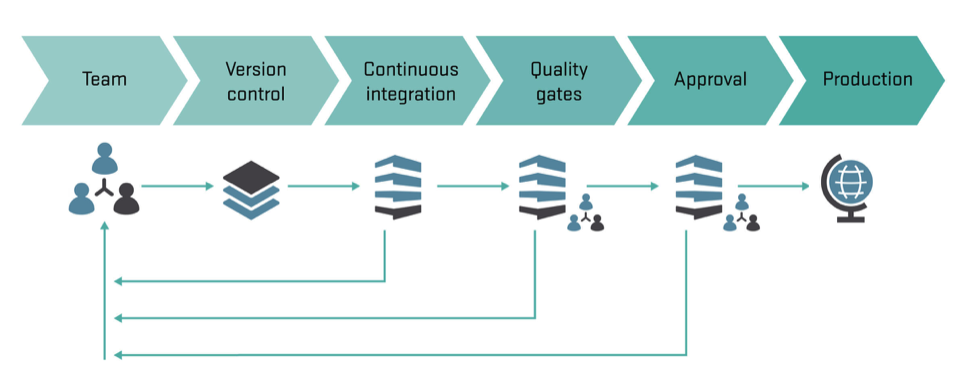
**Continuous Deployment**

**Find A Grave**

# Introduction

**Continuous Delivery (CD) is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time. It is used in software development to automate and improve the process of software delivery.**

# PIpeline for Contituous dePloyment



# benefits of Continious deployment

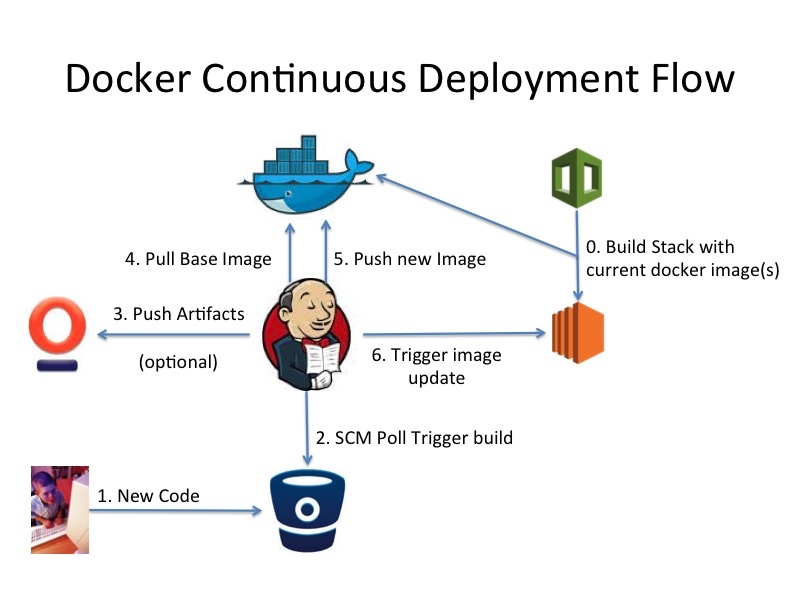
* **Accelerated Time to Market: CD lets an organization deliver the business value inherent in new software releases to customers more quickly. This capability helps the company stay a step ahead of the competition, in today’s competitive economic environment.**
* **Building the Right Product: Frequent releases let the application development teams obtain user feedback more quickly. This lets them work on only the useful features. If they find that a feature isn’t useful, they spend no further effort on it. This helps them build the right product.**
* **Improved Productivity and Efficiency: Significant time savings for developers, testers, operations engineers, etc. through automation.**
* **Reliable Releases: The risks associated with a release have significantly decreased, and the release process has become more reliable. With CD, the deployment process and scripts are tested repeatedly before deployment to production. So, most errors in the deployment process and scripts have already been discovered. With more frequent releases, the number of code changes in each release decreases. This makes finding and fixing any problems that do occur easier, reducing the time in which they have an impact.**

# What is Docker

**Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. Docker uses resource isolation features of the Linux kernel such as cgroups and kernel namespaces to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual** **machines**.

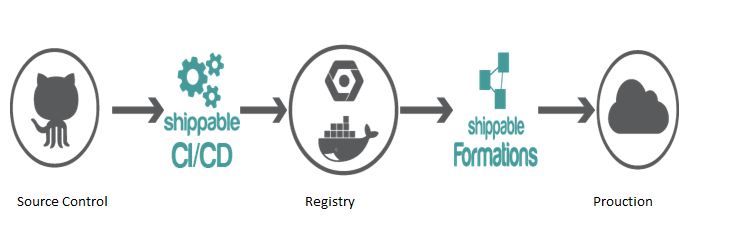
# benefits of using docker in deployment

* **Rapid application deployment – containers include the minimal runtime requirements of the application, reducing their size and allowing them to be deployed quickly.**
* **Portability across machines – an application and all its dependencies can be bundled into a single container that is independent from the host version of Linux kernel, platform distribution, or deployment model. This container can be transferred to another machine that runs Docker, and executed there without compatibility issues.**
* **Version control and component reuse – you can track successive versions of a container, inspect differences, or roll-back to previous versions. Containers reuse components from the preceding layers, which makes them noticeably lightweight.**
* **Sharing – you can use a remote repository to share your container with others. Red Hat provides a registry for this purpose, and it is also possible to configure your own private repository.**
* **Lightweight footprint and minimal overhead – Docker images are typically very small, which facilitates rapid delivery and reduces the time to deploy new application containers.**
* **Simplified maintenance – Docker reduces effort and risk of problems with application dependencies.**

****

# Find A Grave Jenkins and Docker Setup

* **For coninious deployment we follow this Pipeline**



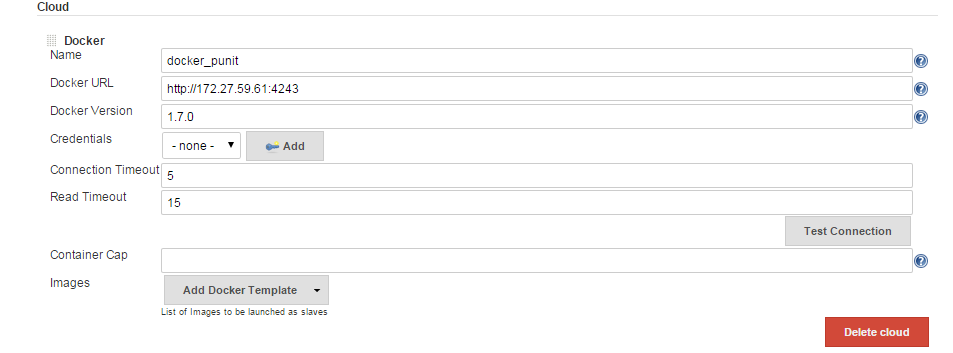
* **First source code of pushed to Source control system (GIT).**
* **This triggers the Jenkins Continuous Server Build which creates platform + Application which are easily deployable on any server.**
* **Then we commit this Artifact and pushed it into the Repository.**
* **This this artifact which contains source code and it’s platform are deployed to the production server.**

# Jenkins Configuration

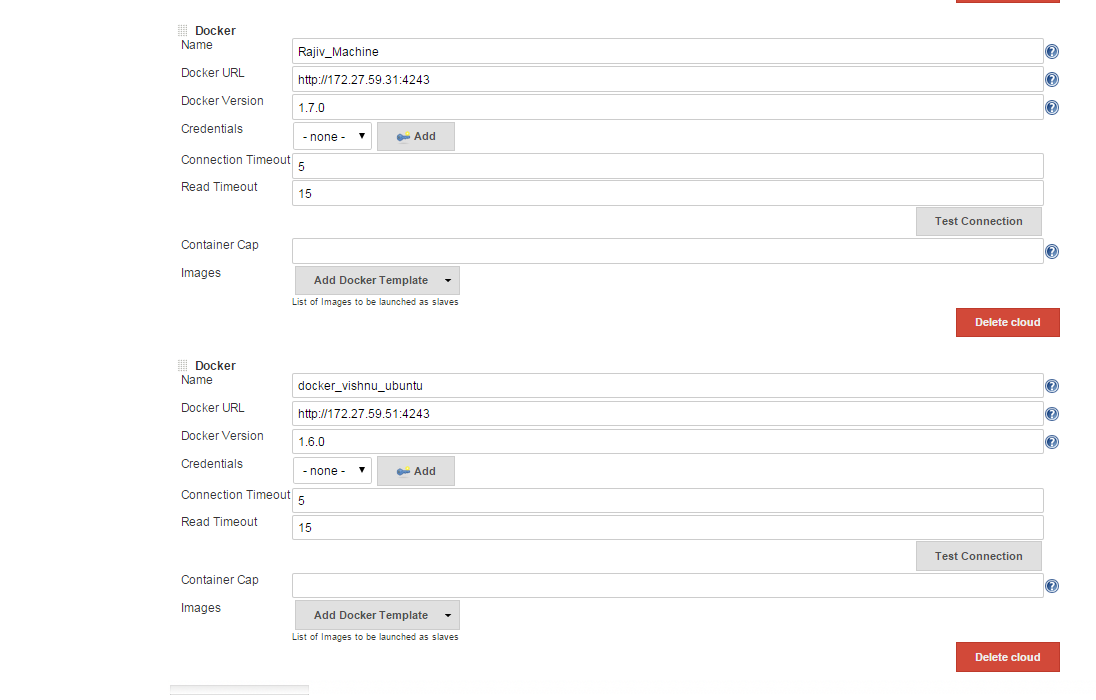
1. **For setting up the entire pipe line we have used various Jenkins plugins and fired various commands on the Execute shell..**
2. **List of plugins which are used in the Project :**
   * GIT
   * Docker Plugin(CD)(It depends on Commons Plugin)
   * Docker Commons Plugin
   * Cloud Bees Docker Custom Build Environment Plugin(For CI)
   * Cloud Bees Docker Build and Publish plugin(For CD)
   * Hudson SCP publisher plugin
   * Email Extension Plugin

# Jenkins Global Configuration

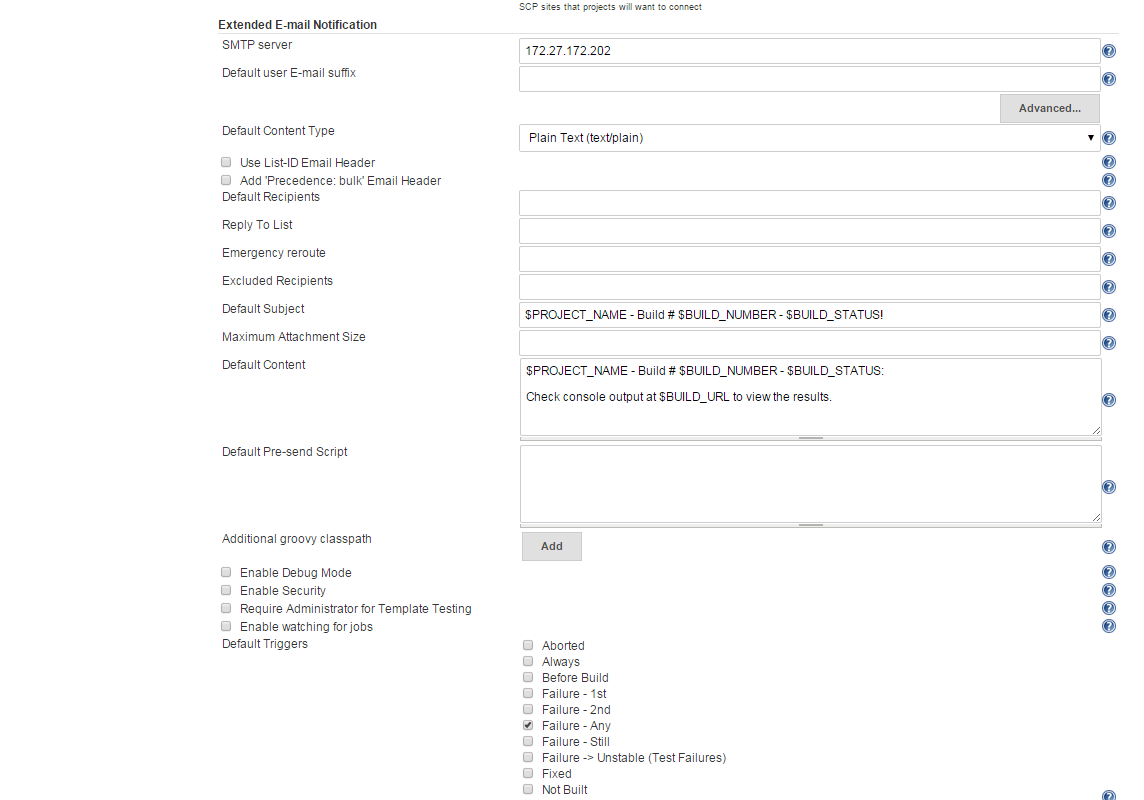
1. **Setting up Docker for Jobs**

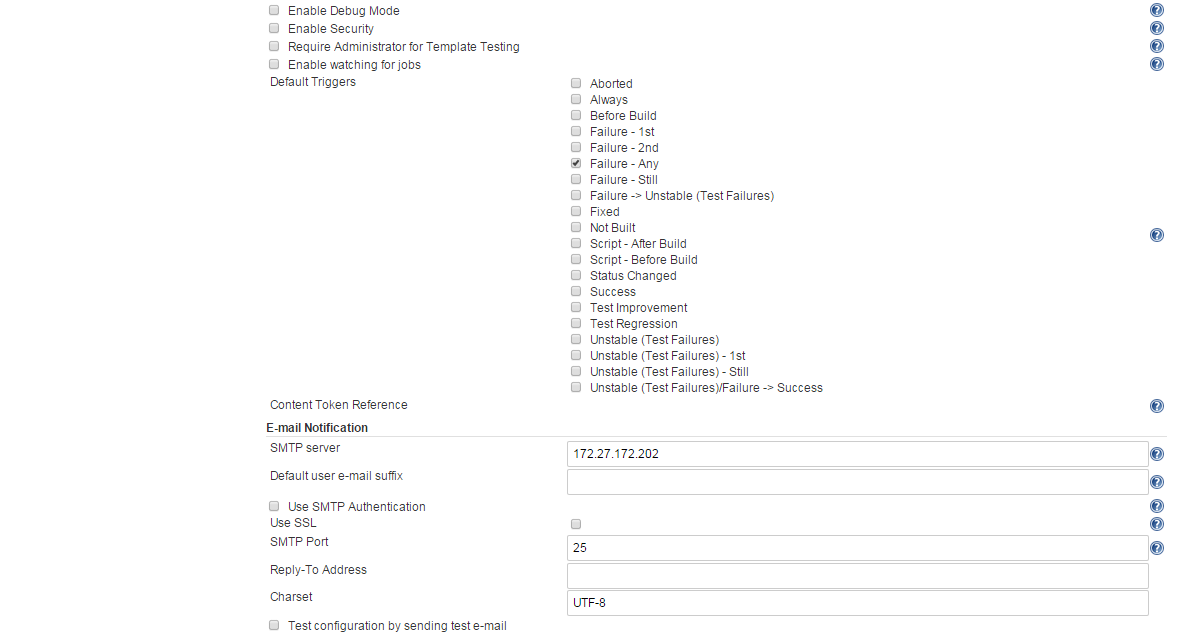
****

* **This Window is enabled when we install Docker Plugin.**
* **Cloud is the details of the machine.**
* **Name : Name of machine where Docker is installed.**
* **Docker URL : URL of the machine where Docker is installed.**
* **Version: Version of Docker.**
* **Click Test connection to verify the connection.**
* **We can add as many number of clouds and use them as and when required**

****

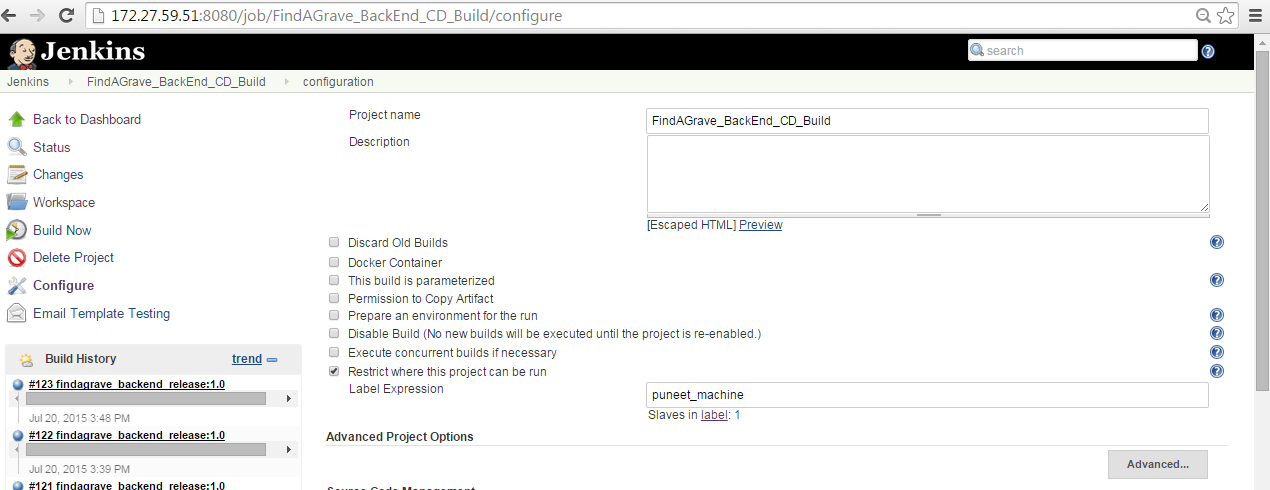
1. **Setting Up email Configuration**

****

****

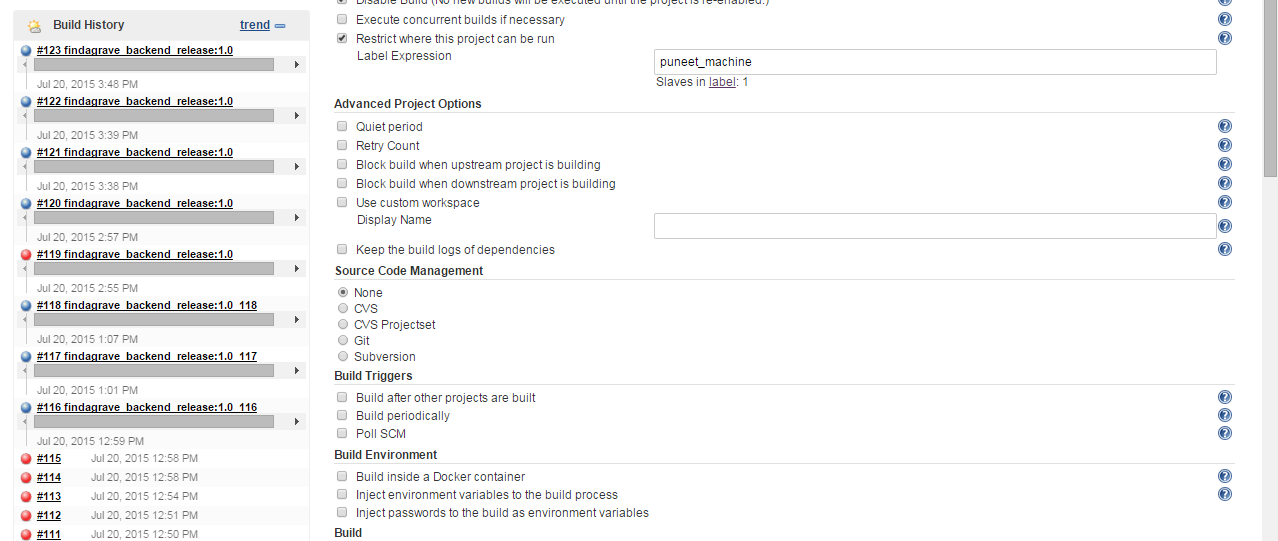
# Jenkins job level configuration

1. **Project name and machine where it runs**

****

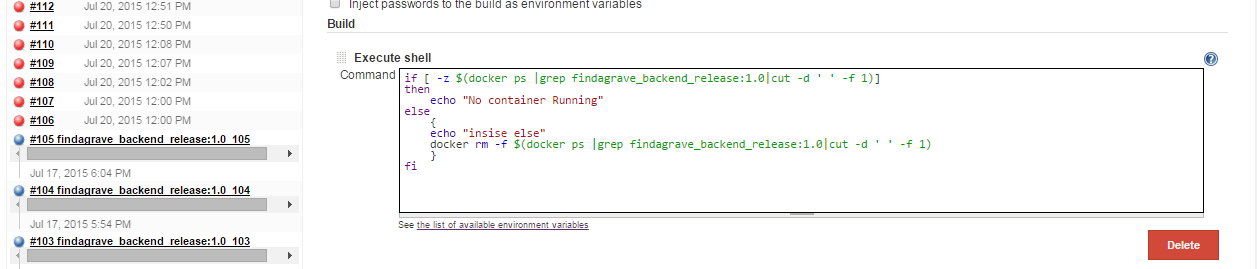
* **Give the desired name of the project .**
* **Restrict where this project can be run (Puneet\_machine) this is the name of the salve which we have added in our Jenkins.**

1. **Source code management**

****

* **We haven’t use any source code management because we copied the desired files and folder to the machine with the help of SCP plugin during the CI process.**
* **The name of the CD Job is same as the name of the folder where we copied the desired files and folders from CI job(after root directory path in the slave).**
* **Once the CD job is triggered it will automatically consider this folder as the Jenkins workspace.**

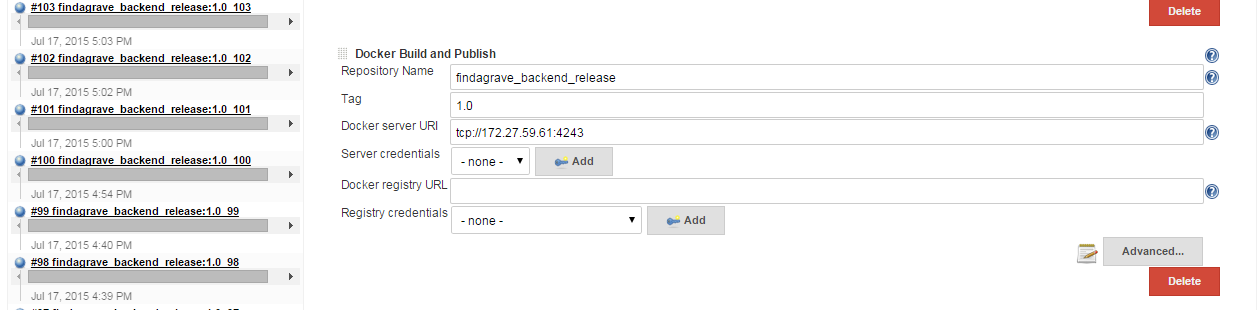
1. **Remove the Existing Running Containers**



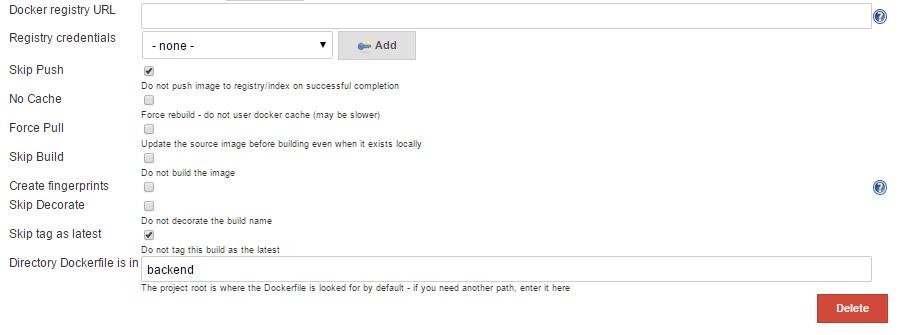
* **In the Build step use Execute shell to remove the existing running containers.**
* **This script echo No Container Running if no containers is running in the system for that job else it will remove the all running containers form the system for that job, It will help us from application port conflicts.**

1. **Creating an Image**

* **This option is enabled when we install Cloud bees Docker build and publish plugin**

****

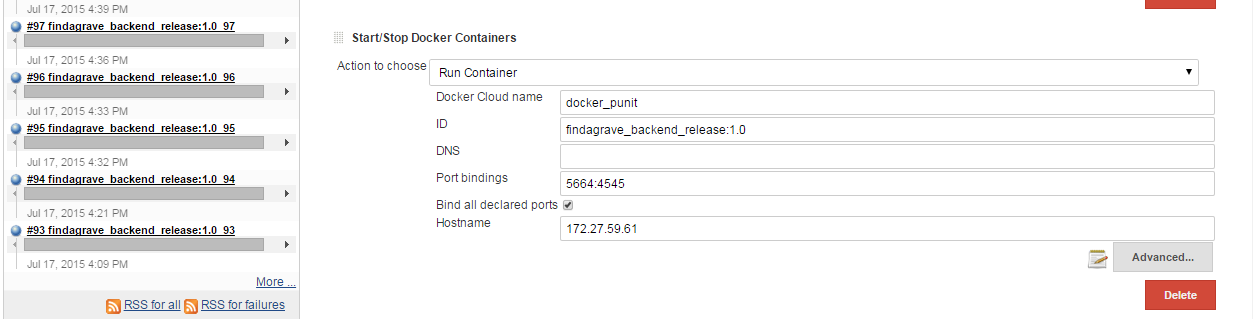
* **Repository Name: Name of the Image Which you want create.**
* **Tag: Provide any tag to the image.**
* **Container server URI: URL of the Docker Host and Port.**



* **Check** the option Skip Push, so that it will not push the image to the hub automatically.
* **Check Skip tag as latest so that Docker will not automatically put the tag as latest.**

1. **Start Stop Docker Container**

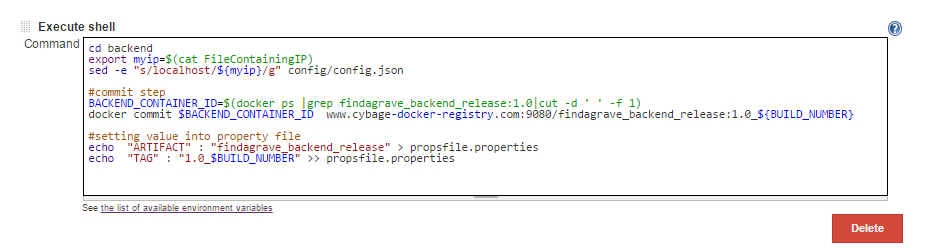
* **This option will be enabled after installing Docker Plugin**

****

* **Action To choose : Run Container.**
* **Docker Cloud Name : Name which we configure in the global configuration.**
* **Port binding : For binding container’s port to the machines port.**
* **Host name : Address of the machine on which Docker is installed.**

1. **Execute shell**

* **To run the Linux commands from Jenkins**

****

* **export myip=$(cat FileContainingIP) : In this we are exporting the local variable in to myip which contains the ip of the DataBase container.**
* **Sed –e “s/localhost/${myip}/g” config/config.json : we are replacing the localhost with database containers IP.**

**Commiting the containers:**

* **BACKEND\_CONTAINER\_ID=$(docker ps | grep findagrave\_backend\_release:1.0|cut –d ‘ ’ –f 1)**

**🡪In this step we are fetching the running containers ID**

* **Docker commit $BACKEND\_CONTAINER\_ID** [**www.cybage-docker-registry.com:9080/findagrave\_backend\_release:1.0\_${BUILD\_NUMBER}**](http://www.cybage-docker-registry.com:9080/findagrave_backend_release:1.0_$%7bBUILD_NUMBER%7d)

**🡪We are commiting the containers with respect to build number.**

**Setting value into property file:**

* **echo "ARTIFACT" : "findagrave\_backend\_release" > propsfile.properties**

**🡪Here we are saving the name of the artifact name into a propsfile.properties**

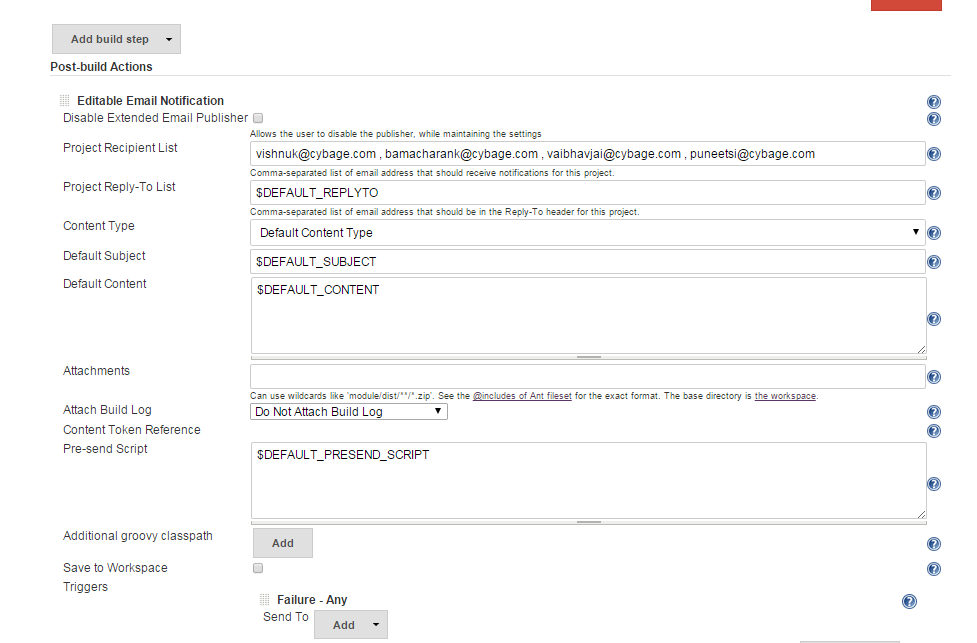
* **echo "TAG" : "1.0\_$BUILD\_NUMBER" >> propsfile.properties**

**🡪Appending the Tag in the props file**

**This over all will help us in pushing the containers into rthe repository.**

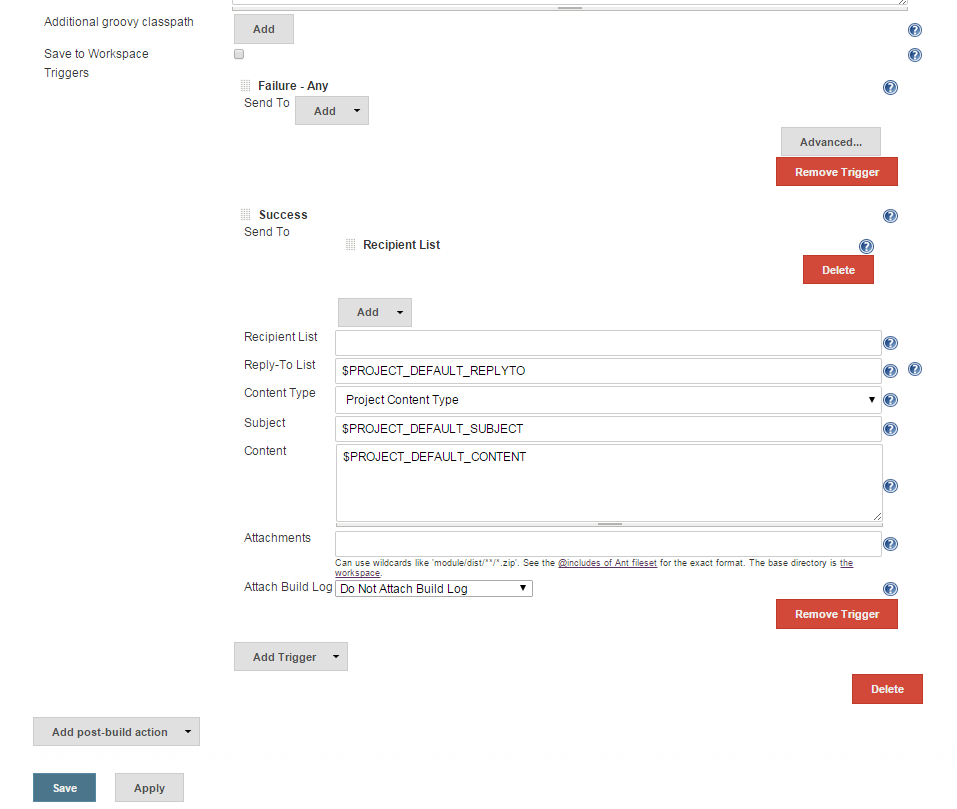
1. **Email Configuration**

**Once the container are up and running , we need to send a mail to QA containig the IP and port of the running container . This feature is provided by Extended email plugin .**

****

**Recipient list is the comma separated list of the receivers who will receive the mail . Default subject and default content can be configured which will have IP and port of the running container.**

**Triggers are added so as to when the mail will be sent. Here the Trigger can be configured on every success or on any failure.**

****

**On successful completion of the build The QA server will have running Frontend and backend containers which can interact with each other and the QA will have IP and port of the running container through which QA will test the application .**